A syntactic component for Vietnamese language processing

Phuong Le-Hong, Azim Roussanaly, Thi Minh Huen Nguyen

A syntactic component for Vietnamese language processing

Phuong Le-Hong[1]*, Azim Roussanaly*[2]*, and Thi Minh Huyen Nguyen*[1]
[1] VNU University of Science, Hanoi, Vietnam
[2] LORIA, Université de Lorraine, Nancy, France

ABSTRACT

This paper presents the development of a grammar and a syntactic parser for the Vietnamese language. We first discuss the construction of a lexicalized tree-adjoining grammar using an automatic extraction approach. We then present the construction and evaluation of a deep syntactic parser based on the extracted grammar. This is a complete system that produces syntactic structures for Vietnamese sentences. A dependency annotation scheme for Vietnamese and an algorithm for extracting dependency structures from derivation trees are also proposed. This is the first Vietnamese parsing system capable of producing both constituency and dependency analyses. It offers encouraging performance: accuracy of 69.33% and 73.21% for constituency and dependency analysis, respectively.

Keywords: language, parsing, segmentation, syntactic component, tagging, tree-adjoining grammar, Vietnamese

1 INTRODUCTION

Natural language processing (NLP) often depends on a syntactic representation of text. Software that can generate such a representation is usually composed of both a grammar and a parser for a given language.

For decades, NLP research has mostly concentrated on English and other well-studied languages. Recently there has been increased interest in languages for which fewer resources exist, notably because of their growing presence on the Internet. Vietnamese, which is among the top 20 most spoken languages (Paul *et al.* 2014), is one such lan-

guage attracting increased attention. Obstacles remain, however, for NLP research in general and grammar development in particular: Vietnamese does not yet have vast and readily available constructed linguistic resources upon which to build effective statistical models, nor does it have reference works upon which new ideas may be experimented.

Moreover, most existing NLP research concerning Vietnamese has been focused on testing the applicability of existing methods and tools developed for English or other Western languages, under the assumption that their logical or statistical well-foundedness might offer cross-language validity; whereas assumptions about the structure of a language are usually made in such tools, and must be amended to adapt them to different linguistic phenomena. For an isolating language such as Vietnamese, techniques developed for inflectional languages cannot be applied "as is".

Our goal is to develop a syntactic parser for the Vietnamese language. We believe that a wide-coverage grammar that incorporates rich statistical information would contribute to the development of basic linguistic resources and tools for automatic processing of Vietnamese written text.

Syntactic parsing is a fundamental task in natural language processing. For Vietnamese, there have been few published works dealing with this problem. This paper presents the construction and evaluation of a deep syntactic parser based on Lexicalized Tree-Adjoining Grammars (LTAG) for the Vietnamese language.

The remainder of the paper is organized as follows. The next section introduces some preliminary concepts of different types of syntactic representation, a brief introduction of the Vietnamese language and the tree-adjoining grammar formalism. Section 3 then presents the construction of a tree-adjoining grammar – the first part of the syntactic component. This grammatical resource is extracted automatically from the Vietnamese treebank. Next, Section 4 discusses the construction of a deep parser based on the extracted grammar. The parser is evaluated in Section 5. Section 6 concludes the paper and suggests some directions for future work.

2 PRELIMINARIES

2.1 *Syntactic representation*

Constituency structure and dependency structure are two types of syntactic representation of a natural language sentence. While a constituency structure represents a nesting of multi-word constituents, a dependency structure represents dependencies between individual words of a sentence. The syntactic dependency represents the fact that the presence of a word is licensed by another word which is its governor. In a typed dependency analysis, grammatical labels are added to the dependencies to mark their grammatical relations, for example *subject* or *indirect object*.

Recently, there have been many published works on dependency analysis for well-studied languages, such as English (Kübler *et al.* 2009) or French (Candito *et al.* 2009b). The dependency parsers developed for these languages are usually probabilistic and trained on corpora available in the language of interest. We can classify the architecture of such parsers into two main types:

- parsers that employ a machine learning method on dependency corpora extracted automatically from treebanks and that directly produce dependency parses (Nivre 2003, McDonald and Pereira 2006, Johansson and Nugues 2008, Candito *et al.* 2010);
- parsers that rely on a sequential process where constituency parses are produced first and then dependency parses are extracted (Candito *et al.* 2009b, de Marneffe *et al.* 2006).

This second type is motivated by the fact that dependency corpora are not readily available for many languages, as in the case of Vietnamese. In such an architecture, we need a module which takes as input constituency parses given by a constituency parser and converts these parses into typed dependency parses as illustrated in Figure 1 and Figure 2 for the English sentence "*A hearing is scheduled on the issue today*" (Nivre and McDonald 2008).

2.2 *A brief overview of Vietnamese*

In this section we present some general characteristics of the Vietnamese language; these are adopted from Hạo (2000), Hữu *et al.* (1998) and Nguyen *et al.* (2006).

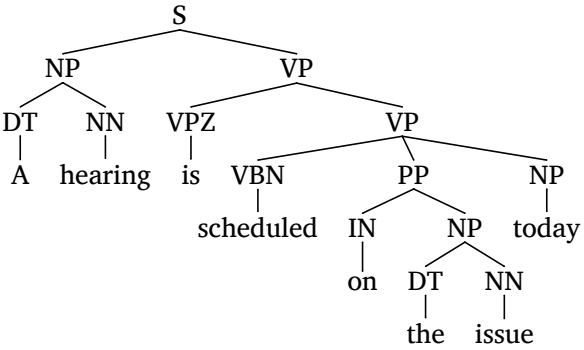


Figure 1: Constituency analysis of an English sentence

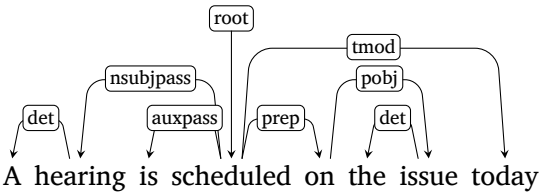


Figure 2: Dependency analysis of an English sentence

Vietnamese belongs to the VietMuong group of the Mon-Khmer branch, which in turn belongs to the Austro-Asiatic language family. Vietnamese is also similar to languages in the Tai family. The Vietnamese vocabulary features a large number of Sino-Vietnamese words which are derived from Chinese (Alves 1999). This vocabulary was originally written with Chinese characters that were used in the Vietnamese writing system, but like all written Vietnamese, is now written with the Latin-based Vietnamese alphabet that was adopted in the early 20th century. Moreover, by being in contact with the French language, Vietnamese was enriched not only in vocabulary but also in syntax by the calque (or loan translation) of French grammar. Thus, for example, the Subject-Verb-Object structure gained prevalence over the natively more common Theme-Rheme construction.

Vietnamese is an isolating language,[1] which means that it is characterized by the following traits:

- it is a monosyllabic language;
- its word forms never change, unlike occidental languages that use morphological variations (e.g. plural form, conjugation);

[1] It is noted that Chinese is also isolating; Chinese is classified in a branch of Sino-Tibetan language family.

- hence, all grammatical relations are manifested by word order and function words.

Vietnamese has a special unit called "tiếng" that corresponds at the same time to a syllable with respect to phonology, a morpheme with respect to morpho-syntax, and a word with respect to sentence constituent creation. For convenience, we call these "tiếng" syllables. The Vietnamese vocabulary contains:

- simple words, which are monosyllabic (e.g. *mưa* (rainy) *nắng* (sunny));
- reduplicated words composed by phonetic reduplication (e.g. *trắng* (*white*) – *trăng trắng* (*whitish*))
- compound words composed by semantic coordination (e.g. *quần* (*trousers*), *áo* (*shirt*) – *quần áo* (*clothes*))
- compound words composed by semantic subordination (e.g. *xe* (*vehicle*), *đạp* (*to pedal*) – *xe đạp* (*bicycle*));
- some compound words whose syllable combination is no longer recognizable (e.g. *bồ nông* (*pelican*))
- complex words phonetically transcribed from foreign languages (e.g. *cà phê* (*coffee*), from the French *café*).

The issue of syntactic category classification for Vietnamese is still in debate in the linguistic community. That lack of consensus is due to the unclear limit between the grammatical roles or syntactic functions of many words as well as the very frequent phenomenon of syntactic category mutation, by which a verb may for example be used as a noun, or even as a preposition. Vietnamese dictionaries (Hoàng 2002) use a set of 8 parts of speech proposed by the Vietnam Committee on Social Sciences (1983).

As for other isolating languages, the most important syntactic information source in Vietnamese is word order. The basic word order is Subject–Verb–Object. There are prepositions but no postpositions. In a noun phrase the main noun precedes the adjectives and the genitive follows the governing noun. These phenomena are subsumed under the term "head-initiality".

The other syntactic means are function words, reduplication, and, in the case of spoken language, intonation.

From the point of view of functional grammar, the syntactic structure of Vietnamese follows a topic-comment structure. It belongs to the class of topic-prominent languages as described by Li and Thompson (1976). In those languages, topics are coded in the surface structure and they tend to control co-referentiality (e.g. *Cây đó lá to nên tôi không thích* (*Tree that leaves big so I not like*), which means *This tree, its leaves are big, so I don't like it*); the topic-oriented "double subject" construction is a basic sentence type (e.g. *Tôi tên là Nam, sinh ở Hà Nội* (*I name be Nam, born in Hanoi*), which means *My name is Nam, I was born in Hanoi*), while such subject-oriented constructions as the passive and "dummy" subject sentences are rare or non-existent (e.g. *There is a cat in the garden* should be translated as *Có một con mèo trong vườn* (*exist one <animal-classifier> cat in garden*)).

2.3 *Tree-adjoining grammars*

In the TAG formalism (Joshi and Schabes 1997), the grammar is defined by a set of elementary trees. A TAG parsing system rewrites nodes of trees rather than symbols of strings as in context-free grammars (CFG). The nodes of these trees are labelled with nonterminals and terminals. Starting from the elementary trees, larger trees are derived using composition operations of substitution and adjunction. In the case of an adjunction, the tree being adjoined has exactly one leaf node that is marked as the foot node (marked with an asterisk). Such a tree is called an *auxiliary tree*. Elementary trees that are not auxiliary trees are called *initial trees*. Each derivation starts with an initial tree. Substituting a tree α in a tree β simply replaces a frontier substitution node in β with α, under the convention that the non-terminal symbol of the substitution node is the same as the root node of α. Only initial trees and derived trees can be substituted in another tree. Adjoining an auxiliary tree β at some node n of a derived tree γ proceeds as follows: the sub-tree t of γ rooted by n is removed from γ, and β is substituted for it instead, where t is substituted in the foot node of β. In the final derived tree, all leaves must have terminal labels.

In TAG, the derived tree does not give enough information to determine how it was constructed. The derivation tree is an object that specifies uniquely how a derived tree was constructed. The root of a derivation tree is labelled by a sentence-type initial tree. All other nodes in the derivation tree are labelled by auxiliary trees in the case

of adjunction or initial trees in the case of substitution. We use the following convention when depicting a derivation tree: trees that are adjoined to their parent tree are linked by a solid line to their parent, and trees that are substituted are linked by a dashed line.

In order to represent natural languages, TAGs are enriched with additional linguistic conventions or principles. First, a TAG for natural languages is *lexicalized* (Schabes 1990), which means that each elementary tree has a lexical anchor (usually unique, but in some cases, there is more than one anchor). Second, the elementary trees of a lexicalized TAG (LTAG) represent extended projections of lexical items (the anchors) and encapsulate all syntactic arguments of the lexical anchor; that is, they contain slots (nonterminal leaves) for all arguments. Furthermore, elementary trees are minimal in the sense that only the arguments of the anchor are encapsulated; all recursion is factored away. This amounts to the *condition on elementary tree minimality* from Frank (2002).

Because of these principles, in linguistic applications, combining two elementary trees corresponds to the application of a predicate to an argument (in case of substitution) or to the addition of modifiers (in case of adjunction). The derivation tree then reflects the predicate-argument structure of the sentence. This is why most approaches to semantics in TAG use the derivation tree as an interface between syntax and semantics.

Figure 3 gives a simple Vietnamese TAG and an analysis of a sentence. The first half of the figure shows the elementary trees of the grammar and the second half shows the derived tree and its corresponding derivation tree, where the notation $<$anchor$>$ represents the elementary tree corresponding to a lexical anchor. A derivation tree in TAG specifies how a derived tree was constructed.

TAG has several advantages over CFG. First, it provides an extended domain of locality. Second, the adjunction operation permits us to model long-distance relationships in single elementary trees due to the factoring of recursion.[2] Third, TAG derivation trees show semantic dependencies between entities in a sentence, as the tree branches rep-

[2] These two properties follow from the mathematical properties of TAGs. TAGs belong to the class of mildly context-sensitive grammars. Context-free languages form a proper subset of tree-adjoining languages (TALs), which in turn form a proper subset of context-sensitive languages.

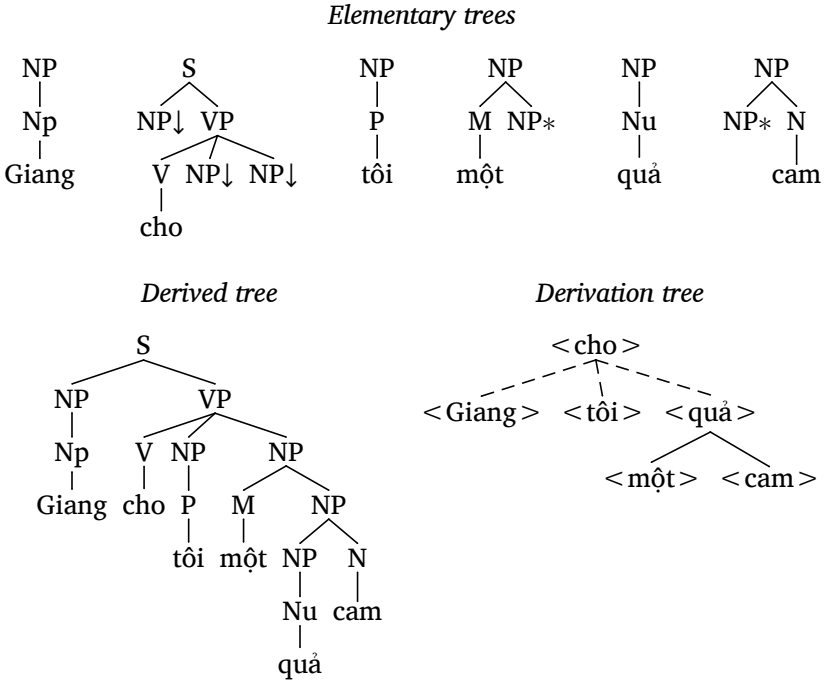


Figure 3: A TAG analysis of the sentence “Giang cho tôi một quả cam” (*Giang gave me an orange*)

resent their combination type (dashed or continuous line for substitution or adjunction, respectively, in Figure 3). In addition, in LTAG, lexical entries naturally capture constraints associated with lexical items, which is not possible in CFG. TAG and LTAG are formally equivalent; however, from the linguistic perspective, LTAG is the system we shall be concerned with in this paper.

3 GRAMMAR EXTRACTION

Since the development of hand-crafted grammars is a time-consuming and labour-intensive task, many studies on automatic and semi-automatic grammar development have been carried out during recent decades. A semi-automatic approach to building a large computational grammar is to rely on a formal language capable of describing the target grammar, e.g. a meta-grammar formalism. Many meta-grammar engineering environments were developed to support the construction

of large computational grammars for natural language. Most of them were used to build large grammars for occidental languages. A typical example is the XMG (eXtensible MetaGrammar) system which supports rapid prototyping of tree-based grammars (Crabbé *et al.* 2013). An alternative approach for obtaining grammars is to extract grammars from a treebank containing syntactically annotated sentences. This is the approach that we chose to rapidly develop a large computational grammar for Vietnamese.

We present in this section a system that automatically extracts lexicalized tree adjoining grammars from treebanks. We first discuss in detail the extraction algorithms and compare them to previous work. We then report the first results for LTAG extraction for Vietnamese, using the recently released Vietnamese treebank.

3.1 *Extracting grammars from treebanks*

There has been much work done on extracting treebank grammars in general and LTAG grammars in particular from annotated corpora, but all of these works are for common languages. Xia *et al.* (2000) and Xia (2001) developed the uniform method of grammar extraction for English, Chinese and Korean. Chiang (2000) developed a system for extracting an LTAG grammar from the English Penn Treebank and used it for statistical parsing with LTAG. Chen and Vijay-Shanker (2000) and Chen *et al.* (2006) extracted TAGs and there are other works based on Chen's approach such as Johansen (2004) and Nasr (2004) for French, and Habash and Rambow (2004) for Arabic. Neumann (2003) extracted lexicalized tree grammars for English from the English Penn Treebank and for German from the NEGRA treebank. Bäcker and Harbusch (2002) extracted an LTAG grammar for German – also from the NEGRA corpus – and used it for supertagging. Kaeshammer (2012) presented a grammar and a lexicon for PLTAG using the German Tiger corpus. Finally, Park (2006) extracted LTAG grammars for Korean from Korean Sejong Treebank.

3.2 *Vietnamese treebank*

Recently, a group of Vietnamese computational linguists has been involved in developing a treebank for Vietnamese (Nguyen *et al.* 2009). This is the treebank we used for our extraction system.

Table 1:
Some Vietnamese treebank tags

No.	Category	Description
1.	S	simple declarative clause
2.	VP	verb phrase
3.	NP	noun phrase
4.	PP	preposition phrase
5.	N	common noun
6.	V	verb
7.	P	pronoun
8.	R	adverb
9.	E	preposition
10.	CC	coordinating conjunction

The construction of a Vietnamese treebank is a branch project of a national project which aims to develop basic resources and tools for Vietnamese language and speech processing.[3] The raw texts of the treebank are collected from the social and political sections of the Youth online daily newspaper. The corpus is divided into three sets corresponding to three annotation levels: word-segmented, POS-tagged and syntax-annotated set. The syntax-annotated corpus, a subset of the POS-tagged set, is currently composed of 10 471 sentences (225 085 tokens). Sentences range from 2 to 105 words, with an average length of 21.75 words. There are 9314 sentences of length 40 words or less. The tagset of the treebank has 38 syntactic labels (18 part-of-speech tags, 17 syntactic category tags, 3 empty categories) and 17 function tags. For details, please refer to Nguyen *et al.* (2009).[4] The meanings of the tags that appear in this paper are listed in Table 1.

3.3 *Extraction algorithms*

In general, our work on extracting an LTAG grammar for Vietnamese follows closely the method of grammar extraction originally proposed by Xia (2001). The extraction process has three steps: first, phrase-structure trees are converted into LTAG derived trees; second, the derived trees are decomposed into a set of elementary trees conforming to their three predefined prototypes; and third, invalid extracted elementary trees are filtered out using linguistic knowledge.

[3] The VLSP project, `http://vlsp.vietlp.org:8080/demo/`.

[4] All the resources are available at the website of the VLSP project.

3.3.1 Building LTAG derived trees

The phrase structures in the Vietnamese treebank follow the English Penn Treebank (PTB) bracketed style format which are not suitable for LTAG extraction due to two reasons. First, the PTB trees do not distinguish heads, arguments and adjuncts as required in derived trees of an LTAG. Second, for each PTB tree, it is not trivial to recover a derivation tree generating it if it is not in a proper format of derived tree.

Therefore, we first have to convert the phrase structures of the treebank into derived trees by augmenting them with additional information needed for extraction.

In this step, we first classify each node in a phrase-structure tree as one of three types: head, argument or modifier. We then build a derived tree by adding intermediate nodes so that at each level of the tree, the nodes satisfy exactly one of the following relations (Xia 2001):

- *predicate-argument relation*: there is one (or more) node(s), where one is the head, and the rest are its arguments;
- *modification relation*: there are exactly two nodes, where one node is modified by the other;
- *coordination relation*: there are exactly three nodes, in which two nodes are coordinated by a conjunction.

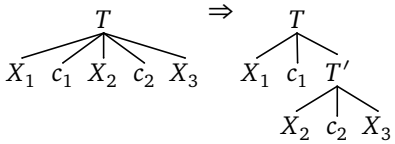
In order to find heads of phrases, we have constructed *a head percolation table* (Magerman 1995; Collins 1997) for the Vietnamese treebank. This table is used to select the head child of a node. In addition, we have also constructed *an argument table* to determine the types of arguments that a head child can take. The argument table helps explicitly mark each sibling of a head child as either an argument or an adjunct according to the tag of the sibling, the tag of the head child, and the position of the sibling with respect to the head child. Together with *the tagset table*, these three tables constitute the Vietnamese treebank-specific information that is required for the extraction algorithms.

Since the conjunction structures are different from the argument and modifier structures, we first recursively bracket all conjunction groups of a treebank tree by Algorithm 1 and then build the full derived tree for the resulting tree by Algorithm 2. A conjunction group

Algorithm 1: ProcessConj(T)

Data: A syntactic tree T.
Result: T whose conjunction groups are processed.
for $K \in T.\texttt{children}$ **do**
    if *IsPhrasal*(K) **then**
        $K \leftarrow$ ProcessConj(K);
$(\mathscr{C}_1, \ldots, \mathscr{C}_k) \leftarrow$ ConjGroups($T.\texttt{children}$);
for $i = 1$ *to* k **do**
    if $\|\mathscr{C}_i\| > 1$ **then**
        InsertNode($T, \mathscr{C}_i$);
if $k > 2$ **then**
    for $i = k$ ***downto*** 3 **do**
        $\mathscr{L} \leftarrow \mathscr{C}_{i-1} \cup c_{i-1} \cup \mathscr{C}_i$;
        $T' \leftarrow$ InsertNode($T, \mathscr{L}$);
        $\mathscr{C}_{i-1} \leftarrow T'$;
return T;

Figure 4: Transformation of conjunction groups



is a group of coordinating words or phrases connected by one or more coordinating conjunction. The form of a conjunction group is either "A and B" or "A or B".[5] Figure 4 shows a tree with conjunction groups before and after being processed by Algorithm 1 where c_i are coordinating conjunctions and X_i are conjunction groups. Figure 5 shows a realisation of Algorithm 2 where A_i are arguments of the head child H of T and M_i are modifiers of H. These two algorithms use the function InsertNode($T, \mathscr{L}$) to insert an intermediate node between a node T and a list of its child nodes $\mathscr{L}$. This new node is a child of T, has the same label as T and has $\mathscr{L}$ as the list of its children. The function IsPhrasal(X) checks whether X is a phrasal node or not.[6] The function

[5] In the treebank, there are no conjunctions which use the coordinating punctuation; that is, a structure like "A, B and C" is not present.

[6] A phrasal node is defined to be a node which is not a leaf or a preterminal. This means that it must have two or more children, or one child that is not a leaf.

Algorithm 2: BuildDerivedTree(T)

Data: A tree T whose conjunction groups have been processed.
Result: A derived tree whose root is T.
if *(not IsPhrasal*(T)*)* **then**
  return T;
$H \leftarrow$ HeadChild(T);
if *not IsLeaf*(H) **then**
  for $K \in T$.`children` **do**
    $K \leftarrow$ BuildDerivedTree(K);
  $\mathscr{A} \leftarrow$ ArgNodes($H, \mathscr{L}$);
  $\mathscr{M} \leftarrow$ ModNodes($H, \mathscr{L}$);
  $m \leftarrow \|\mathscr{M}\|$;
  if $m > 0$ **then**
    $\mathscr{L} \leftarrow \{H\} \cup \mathscr{A}$;
    $T' \leftarrow$ InsertNode($T, \mathscr{L}$);
  $(M_1, M_2, \ldots, M_m) \leftarrow \mathscr{M}$;
  for $i \leftarrow 1$ ***to*** $m-1$ **do**
    $\mathscr{L} \leftarrow \{M_i, T'\}$;
    $T'' \leftarrow$ InsertNode($T, \mathscr{L}$);
    $T' \leftarrow T''$;
return T;

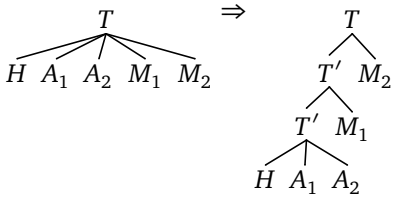


Figure 5: An example of derived tree realisation

ConjGroups($\mathscr{L}$) returns k groups of components $\mathscr{C}_i$ of $\mathscr{L}$ which are separated by $k-1$ conjunctions $c_1, \ldots, c_{k-1}$, which have a special POS tag in the treebank (CC).

Algorithm 2 uses several simple functions. The HeadChild(X) function selects the head child of a node X according to a head percolation table. The function IsLeaf(X) checks whether a node X is a

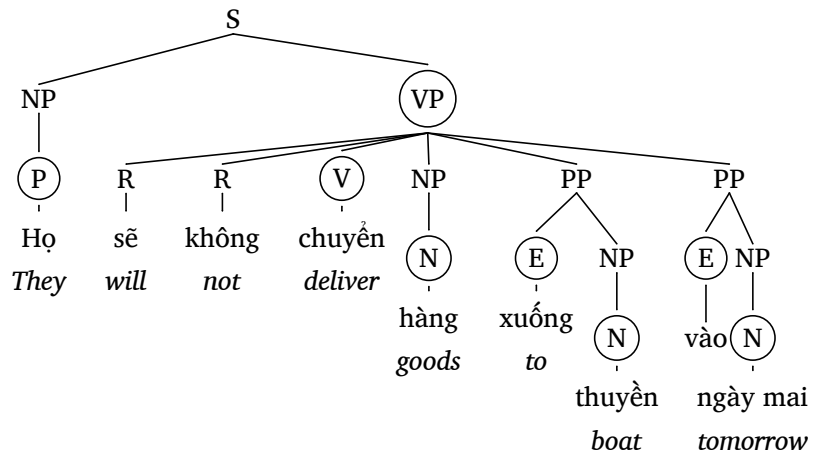


Figure 6: A parse tree of the Vietnamese treebank

leaf node or not. The functions ArgNodes($H, \mathscr{L}$) and ModNodes($H, \mathscr{L}$) each return a list of nodes which are arguments and modifiers, respectively, of a node H. The list $\mathscr{L}$ contains all sisters of H.

For example, Figure 6 shows the phrase structure of a sentence extracted from the Vietnamese treebank "*Họ sẽ không chuyển hàng xuống thuyền vào ngày mai.*" (They will not deliver the goods to the boat tomorrow.) The head children of phrases are circled.

The derived tree of the sentence once processed by Algorithm 2 is shown in Figure 7, wherein the inserted nodes are marked by the quotation mark symbol (').

3.3.2 Building elementary trees

At this step, each derived tree is decomposed into a set of elementary trees. The recursive structures of the derived tree are factored out and will become auxiliary trees, and the remaining non-recursive structures will be extracted as initial trees.

Extracted elementary trees fall into one of three prototypes as determined by the relation between the anchor and other nodes, as shown in Figure 8. The extraction process involves copying nodes from the derived tree for building elementary trees. The result of the extraction process is three sets of elementary trees: $\mathscr{S}$ contains spine trees, $\mathscr{M}$ contains modifier trees and $\mathscr{C}$ contains conjunction trees.

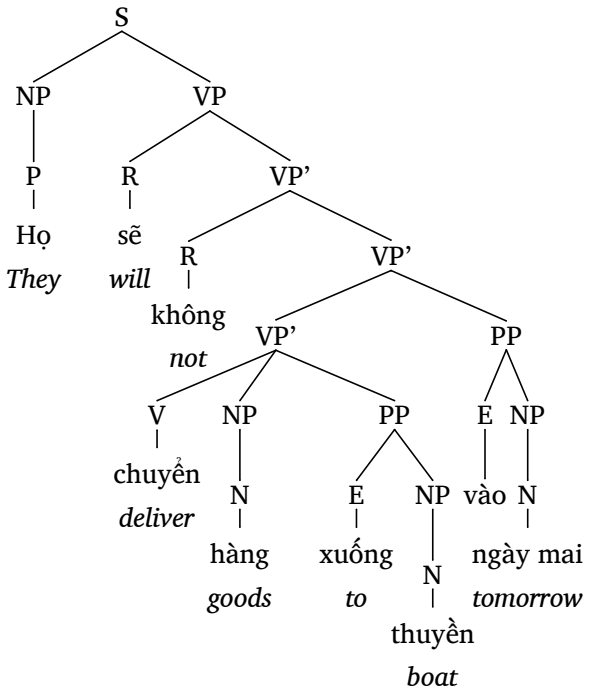


Figure 7:
The derived tree of the treebank tree in Figure 6

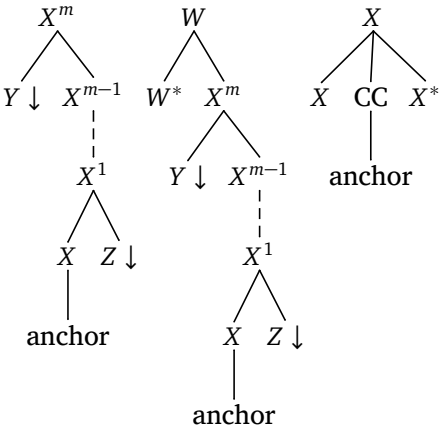


Figure 8:
Prototypes of spine trees and auxiliary trees

Algorithm 3: BuildElementaryTrees(T)

```
Data: T is a derived tree.
Result: Sets 𝒮, ℳ, 𝒞 of elementary trees.
if (not IsPhrasal(T)) then
    return ;
{H_0, H_1, ..., H_n} ← HeadPath(T);
ok ← false;
P ← H_0;
for j ← 1 to n do
    ℒ ← Sisters(H_j);
    if |ℒ| > 0 then
        Rel ← GetRelation(H_j, ℒ);
        if Rel = Coordination then
            𝒞 ← 𝒞 ∪ BuildConjTree(P);
        if Rel = Modification then
            ℳ ← ℳ ∪ BuildModTree(P);
            if j = 1 then
                𝒮 ← 𝒮 ∪ BuildSpineTree(P);
                ok ← true;
        if Rel = Argument then
            if not ok and not IsLinkNode(P) then
                𝒮 ← 𝒮 ∪ BuildSpineTree(P);
                ok ← true;
        else
            if not IsLinkNode(P) and IsPhrasal(P) then
            𝒮 ← 𝒮 ∪ BuildSpineTree(P) ;
        P ← H_j;
```

To build elementary trees from a derived tree T, we first find the head path[7] $\{H_0, H_1, \ldots, H_n\}$ of T. For each parent P and its head child H, we get the list $\mathscr{L}$ of sisters of H and determine the relation between H and $\mathscr{L}$. If the relation is coordination, a conjunction tree will

[7] A *head path* starting from a node T in a derived tree is the unique path from T to a leaf node where each node except T is the head child of its parents. Here $H_0 \equiv T$ and H_j is the parent of its head child H_{j+1}. A node on the head path is called a *link node* if its label is the same as that of its parent.

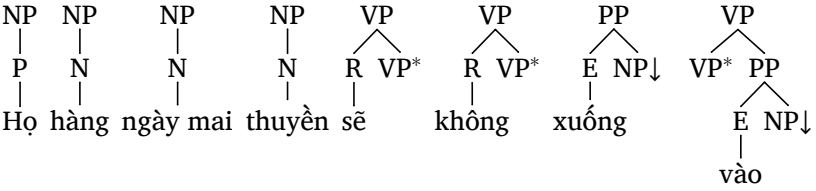


Figure 9: Extracted elementary trees

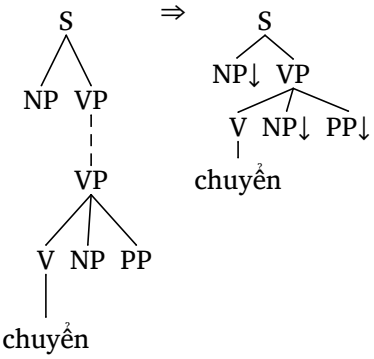


Figure 10: Merge link nodes to get a spine tree

be extracted; if the relation is modification, a modifier tree will be extracted; otherwise, the relation is predicate-argument and a spine tree will be extracted. Algorithm 3 shows the complete extraction algorithm. This algorithm uses additional functions as follows:

- BuildSpineTree(T) which creates a spine tree;
- MergeLinkNodes(T) which merges all link nodes of a spine tree into one node (see Figure 10 for an example);
- BuildModTree(T) which creates a modifier tree;
- BuildConjTree(T) which creates a conjunction tree.

As an example, from the derived tree shown in Figure 7, nine trees are extracted by algorithms as shown in Figure 9 and Figure 10.

3.3.3 Filtering out invalid trees

Annotation errors may be present in any particular treebank. The errors in parse trees will result in incorrect elementary trees. An elementary tree is called invalid if it does not satisfy some linguistic requirement. We have constructed some linguistic rules for filtering out invalid elementary trees. For example, in Vietnamese, an adjective (or an adjectival phrase) can be an argument of a noun (or a noun phrase); however, it must always be to the right of the noun. For instance, in

Algorithm 4: BuildSpineTree(T)

Data: T is a derived tree.
Result: A spine tree.
$T_c \leftarrow$ Copy(T);
$P \leftarrow T_c$;
$H \leftarrow$ `NULL`;
repeat
  $H \leftarrow$ HeadChild(P);
  $\mathscr{L} \leftarrow$ Sisters(H);
  if $|\mathscr{L}| > 0$ **then**
    `Rel` $\leftarrow$ GetRelation($H, \mathscr{L}$);
    if `Rel` = `Argument` **then**
      for $A \in \mathscr{L}$ **do**
        BuildElementaryTrees(A);
        A.`children` $\leftarrow \emptyset$;
        A.`type` $\leftarrow$ `Substitution`;
    else
      for $A \in \mathscr{L}$ **do**
        P.`children` $\leftarrow P$.`children` $\setminus A$;
  $P \leftarrow H$;
until *(*$H =$ `NULL`*)*;
return MergeLinkNodes(T_c);

Algorithm 5: BuildModTree(T)

Data: T is a derived tree
Result: a modifier tree
$T_c \leftarrow$ Copy(T);
$H \leftarrow$ HeadChild(T_c);
H.`children` $\leftarrow \emptyset$;
H.`type` $\leftarrow$ `Foot`;
$M \leftarrow$ Modifier(H);
$T' \leftarrow$ BuildSpineTree(M);
if $|M$.`children`$| > 1$ **then**
  BuildElementaryTrees(M);
$M \leftarrow T'$;
return T_c;

Algorithm 6: BuildConjTree(T)

Data: T is a derived tree.
Result: A conjunction tree.
$T_c \leftarrow$ Copy(T);
$H \leftarrow$ HeadChild(T_c);
BuildElementaryTrees(H);
$K \leftarrow$ Coordinator(H);
BuildElementaryTrees(K);
H.`children` $\leftarrow \emptyset$;
H.`type` $\leftarrow$ `Foot`;
K.`children` $\leftarrow \emptyset$;
K.`type` $\leftarrow$ `Substitution`;
return T_c;

the noun phrase *cô gái đẹp* (*beautiful girl*), the adjective *đẹp* (*beautiful*) must go after the noun *cô gái* (*girl*). Thus if there is an adjective on the left of a noun of an extracted spine tree, the tree is invalid and it must be filtered out.

3.4 *Comparison with previous work*

As mentioned above, our approach for LTAG extraction follows the uniform method of grammar extraction proposed by Xia (2001). Nevertheless, there are some differences between our design and implementation of extraction algorithms and that of Xia.

First, in the step in which we build the derived tree, we first recursively bracket all conjunction groups of the tree before fully bracketing the arguments and modifiers of the resulting tree. We think that this approach is easier to understand and implement since conjunction structures are different from argument and modifier structures. Second, in the elementary tree decomposition step, we do not split each node in the derived tree into the top and bottom parts as was done in Xia's approach of Xia. In our implementation, the nodes are directly copied to build extracted trees. Third, the tree extraction process is separated into functions; each function builds a particular type of elementary tree; and these functions can call each other to repeat the extraction process for the subtrees whose roots are not yet visited. In spite of using recursive functions, our extraction algorithms are carefully designed to avoid redundant or repeating function calls:

Table 2: Some tags in the Vietnamese treebank tagset are merged into a single tag

Category	Original tags	Tags in G_2
noun phrases	NP/WHNP	NP
adjective phrases	AP/WHAP	AP
adverbial phrases	RP/WHRP	RP
preposition phrases	PP/WHPP	PP
clauses	S/SQ	S

Table 3: Two LTAG grammars extracted from the Vietnamese treebank

Type	# of trees	# of templates
G_1	**46 382**	**2317**
Spine trees	24 973	1022
Modifier trees	21 309	1223
Conjunction trees	100	72
G_2	**46 102**	**2113**
Spine trees	24 884	952
Modifier trees	21 121	1093
Conjunction trees	97	68

each node is assured to be visited one time. The "divide and conquer" approach seems to be reasonably efficient and is easy to optimise.

3.5 *An LTAG for Vietnamese*

We ran extraction algorithms on the Vietnamese treebank and extracted two treebank grammars. The first one, G_1, uses the original tagset of the treebank. The second one, G_2, uses a reduced tagset, where some sets of tags in the treebank are consolidated, as shown in Table 2. The grammar G_2 is smaller than G_1 and it is presumed that the sparse data problem is less severe when G_2 is used.

We count the number of elementary trees and tree templates. The sizes of the two grammars are in Table 3. Recall that a template is an elementary tree without the anchor word.

There are 15 035 unique words in the treebank and the average number of elementary trees that a word anchors is around 3.07. We also count the number of context-free rules of the grammars where the rules are simply read off the templates in an extracted LTAG. The extracted grammars G_1 and G_2 have 851 and 727 context-free rules, respectively.

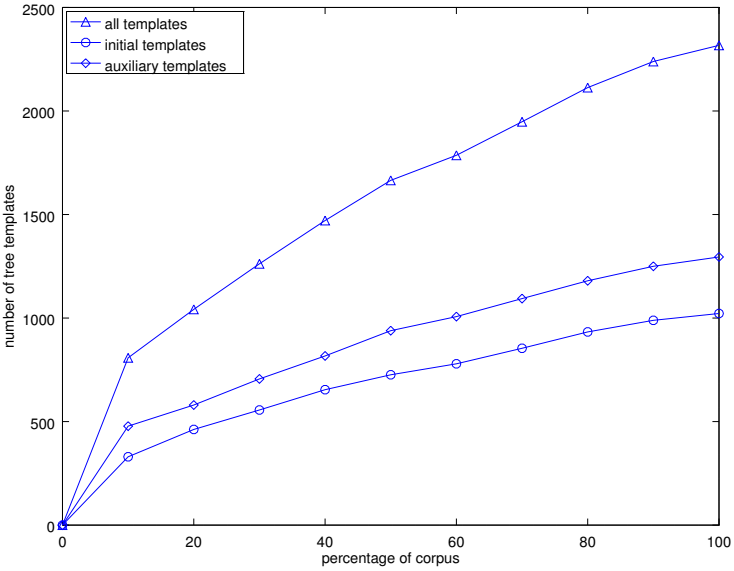


Figure 11: The growth of tree templates

In order to evaluate the coverage of the Vietnamese treebank, we count the number of extracted tree templates with respect to the size of the treebank. Figure 11 shows that the number of templates converges very slowly as the size of the corpus grows, implying that there are many unseen templates. This experiment also implies that the size of the current Vietnamese treebank is not large enough to cover all the grammatical templates of the Vietnamese language.

We have developed a software package[8] that implements the presented algorithms for extracting an LTAG for Vietnamese. The software is written in the Java programming language and is freely distributed under the GNU/GPL license. The software is very efficient in term of extraction speed: it takes only 165 seconds to extract the entire grammar G_1 on an ordinary personal computer.[9] It should be straightforward to extend the software in order to extract LTAGs from treebanks of other languages since the language-specific information is intentionally factored out of the general framework. In order to use the software on a treebank of a given language, a user would need to provide the treebank-specific information for that language: a tagset, a head percolation table, and an argument table.

[8] `http://mim.hus.vnu.edu.vn/phuonglh/softwares/vnLExtractor`

[9] On an Intel Core 2 Duo CPU U9600 with 4GB RAM.

3.6 *Summary*

In this section, we have presented a system that automatically extracts LTAGs from treebanks. The system has been used to extract an LTAG for the Vietnamese language from the recently released Vietnamese treebank. The extracted Vietnamese LTAG covers the corpus; that is, the corpus can be seen as a collection of derived trees for the grammar and can be used to train statistical LTAG parsers directly.

The number of templates extracted from the current Vietnamese treebank converges slowly. This implies that there are many new templates outside the corpus and the current Vietnamese treebank is not large nor typical enough to cover all the grammatical templates of the Vietnamese language.

We are currently experimenting with extracting a French LTAG from a French treebank (Abeillé *et al.* 2003). We also plan to compare quantitatively syntactic structures of French and Vietnamese. We believe that a quantitative comparison of the two grammars may reveal interesting relations between them.

4 PARSER CONSTRUCTION

We present in this section the construction of a deep syntactic parser for Vietnamese. Our parser is able to produce both constituency and dependency analyses for a given sentence.

4.1 *Preprocessing pipeline*

Before being parsed, a text is fed to a chain of preprocessing modules including a sentence segmenter, a word tokenizer and a tagger. In particular, we have integrated the following preprocessing modules into the parser:

- **vnSentDetector** – a sentence detector which segments a text into sentences;
- **vnTokenizer** – a tokenizer which segments sentences into words or lexical units (Le-Hong *et al.* 2008);
- **vnTagger** – a part-of-speech tagger which tags each word of a sentence with its most appropriate syntactic category (Le-Hong *et al.* 2010).

We have adapted an LTAG parser developed at the LORIA[10] laboratory to construct a deep syntactic parser for Vietnamese. This parser was initially used to parse French text (Roussanaly *et al.* 2005). Given a sentence, the parser outputs all possible constituency parses and their corresponding derivation trees. The most important improvement we made to the parser is the refactoring and introduction of general interfaces and modules for preprocessing tasks (sentence detection, word segmentation, POS tagging) which naturally depend on specific languages. We have also enriched the parser by adding a supplementary module which extracts dependency parses from constituency parses given by the parser.[11] This module implements the dependency analysis extraction algorithm which will be described in the next subsections.

4.2 *Dependency annotation schema*

There exist many schema for dependency annotation. Examples include the Stanford Dependency (SD) annotation scheme (de Marneffe *et al.* 2006), created via an automated conversion of the English Penn Treebank; the PARC 700 scheme (King *et al.* 2003), inspired by functional structures of lexical functional grammars; and the GR scheme (Caroll *et al.* 1998) or EASy (Paroubek *et al.* 2005) for French. Recently, McDonald *et al.* (2013) presented a universal treebank with homogeneous syntactic dependency annotation for six languages: German, English, Swedish, Spanish, French and Korean. The multiplicity of these different annotation schema is due to different linguistic and practical choices. We prefer defining an annotation scheme of surface dependency for the Vietnamese language which can be not only convertible to different standards cited above but also enlargeable to finer dependency schema if necessary. The current scheme contains 13 grammatical relations representing principal functional dependencies between Vietnamese words. All these dependencies use the syntactic categories defined in the Vietnamese treebank (Nguyen *et al.* 2009) and they are divided into three groups.

The first group, *arg*, represents the relationship between a head word and its argument. There are two types of arguments: subject

[10] `http://www.loria.fr/`

[11] `http://mim.hus.vnu.edu.vn/phuonglh/softwares/vnLTAGParser`

(*subj*) or object (*obj*). It is worth noting that Vietnamese is a topic-prominent language where sentences are structured around topics rather than subjects and objects. In many cases, we cannot identify the subject and the object of a Vietnamese sentence by their respective positions. The distinction between subject and object of a Vietnamese sentence is thus not a trivial task, expecially in an automatic process. Therefore, at the moment, we do not distinguish the two relations *subj* and *obj* in our evaluations. The second group, *mod*, represents modification relations of a word and its head word (or its governor). According to the syntactic category of the modifier, we distinguish nine modification relations named *modN* (nominal modifier), *modM* (numeral modifier), *modA* (adjective modifier), *modR* (adverbial modifier), *modE* (prepositional modifier), *modV* (verbal modifier), *modL* (determinant modifier), *modP* (pronominal modifier) and *modC* (subordinating coordination modifier). The third group, *coord*, represents dependencies of each lexical head of two coordinating phrases on the conjunction.

Having defined a dependency annotation scheme for Vietnamese, we now propose an algorithm for automatically extracting dependency analyses from TAG derivation trees.

4.3 *Dependency relation extraction*

It has been shown that the TAG formalism shares many important similarities with the dependency grammar formalism (Rambow and Joshi 1994). A derivation tree of TAG can be converted superficially into a dependency tree in the case of lexicalized grammars (Kallmeyer and Kuhlmann 2012). The main idea is to transform each derivation operation into a dependency relation. A derivation operation between a source tree t_1 and a target tree t_2 results in a dependency relation between the head word of t_1 as governor and the head of t_2 as dependent word.

The dependency analysis corresponding to the analysis in Figure 3 is shown in Figure 12. We see that the derivation tree can be transformed into the dependency tree by a simple transformation in which each node of the derivation tree (representing an elementary tree) is replaced with its lexical node. Here, we want to extract typed dependencies where each one is labelled by a grammatical relation following the annotation scheme defined above. We thus need to consider the

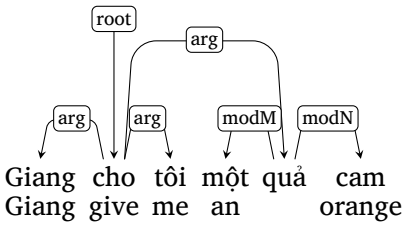


Figure 12: Dependency tree corresponding to the analysis in Figure 3

operation done at each node of the derivation tree. If it is a substitution, a relation of type *arg* will be created; if it is an adjunction, a relation of type *mod* will be created and its label can be determined by examining the syntactic category of the concerned word at the lexical node of the derivation tree.

Figure 13: Examples of coordination auxiliary trees

The most difficult case is the construction of coordination relations where we must consider three related nodes and two combination operations at the same time since an auxiliary tree for conjunctions in TAG has a specific form having a substitution node and a foot node, as illustrated in example trees in Figure 13. We propose an algorithm for the automatic extraction of dependency relations from a derivation tree given by a constituency parser. The algorithm ExtractRelations(N) (Algorithm 7) shows the extraction procedure in detail.

This algorithm uses some supplementary functions as follows. The function LexicalNode(N) returns the lexical head of a node of an input derivation tree N, while the function POSNode(N) returns the part-of-speech of a lexical head. The functions IsSubst() and IsAdj() are called at each node of the derivation tree to verify whether the node is about a substitution or an adjunction. Finally, the function NewRelation($\texttt{type}, w_1, w_2$) creates and returns a new relation of type `type` between two lexical units w_1 and w_2.

Algorithm 7: ExtractRelations(N)

Data: A derivation tree N.
Result: A set $\mathscr{R}$ of dependency relations.
$w_n \leftarrow$ LexicalNode(N);
$t_n \leftarrow$ POSNode(N);
for $K \in N.\texttt{children}$ **do**
  $w_k \leftarrow$ LexicalNode(K);
  $t_k \leftarrow$ POSNode(K);
  if $K.IsSubst()$ **then**
    if $t_n = \texttt{CC}$ **then**
      $\mathscr{R} \leftarrow \mathscr{R} \cup$ NewRelation($\texttt{coord}, w_n, w_k$);
    else
      $\mathscr{R} \leftarrow \mathscr{R} \cup$ NewRelation($\texttt{arg}, w_n, w_k$);
  else
    if $K.IsAdj()$ **then**
      if $t_k \in \{\texttt{A}, \texttt{N}, \texttt{R}, \texttt{V}, \texttt{E}, \texttt{L}, \texttt{M}, \texttt{P}, \texttt{C}\}$ **then**
        $\mathscr{R} \leftarrow \mathscr{R} \cup$ NewRelation($\texttt{mod}t_k, w_n, w_k$);
      if $t_k = \texttt{CC}$ **then**
        $\mathscr{R} \leftarrow \mathscr{R} \cup$ NewRelation($\texttt{coord}, w_k, w_n$);
  // *Recursively extract relations from tree* K;
  ExtractRelations(K);
return $\mathscr{R}$;

For example, the application of this algorithm on the input derivation tree in Figure 1 results in the following relations: *arg*(cho,Giang), *arg*(cho,tôi), *arg*(cho,quả), *modM*(quả,một), *modN*(quả,cam).

5 PARSER EVALUATION

In this section, we evaluate the parser on a test corpus. The parser performance is considered in two versions, with and without using part-of-speech (POS) tagging.

The grammar used to evaluate the parser is an LTAG extracted from the Vietnamese Treebank (Nguyen *et al.* 2009) containing 10 163 sentences (225 085 words, about 22.14 words per sentence on average). Figure 14 shows the distribution of the number of sentences ac-

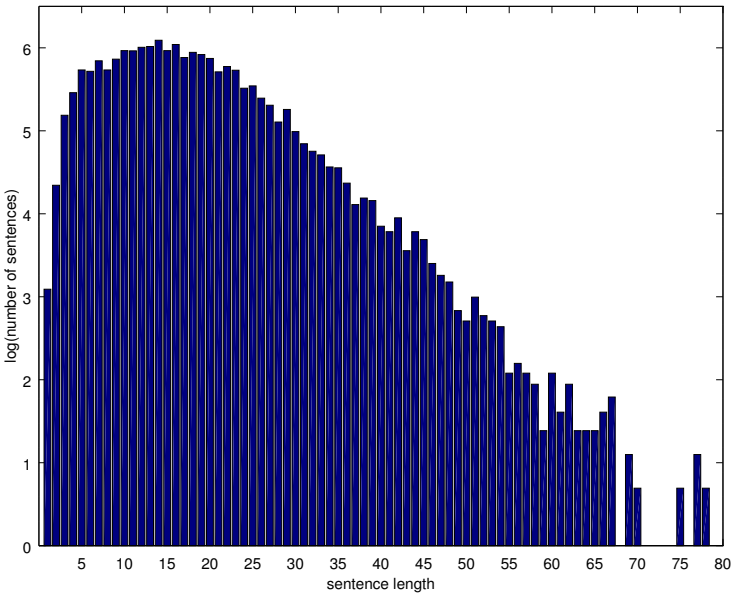


Figure 14:
The distribution of the number of sentences according to their length

cording to their lengths. We see that most of the sentences have a length between 5 and 30 words.

We choose a subset of the treebank containing 8808 sentences of length 30 words or less as an evaluation corpus. This corpus is divided into two sets: a training set (95% of the corpus, 8367 sentences) and a test set (5% of the corpus, 441 sentences). We use **vnLExtractor** to extract an LTAG for Vietnamese from the training set. This grammar contains 35 655 elementary trees instantiated from 1658 tree templates. The size of this grammar is shown in Table 4.

Type	**Number of trees**	**Number of templates**
Spine trees	19 708	741
Modifier trees	15 868	860
Conjunction trees	79	57
Total	**35 655**	**1658**

Table 4:
Size of the LTAG extracted from the training corpus

To evaluate the parser, we make use of two measures: *tree accuracy* (or T-accuracy) and *dependency accuracy* (or D-accuracy).[12]

[12] In computing these scores, unanalyzable sentences and punctuations are not taken into account.

Table 5: Performance of the constituency analysis without or with POS tagging

T-accuracy	All		≤ 10 words	
	No POS	POS	No POS	POS
Precision	67.98	69.15	71.28	71.60
Recall	68.40	69.52	71.39	72.30
F-measure	68.19	**69.33**	71.33	**71.95**
Complete match	13.00	16.67	17.57	20.69
Average crossing	2.66	2.39	1.80	1.69
No crossing	23.00	27.78	29.73	32.76
Fewer than three crossings	55.00	54.17	68.92	65.52
Tagging accuracy	87.72	95.25	87.34	95.43

Table 6: Performance of the dependency analysis without or with POS tagging

D-accuracy	With type		Without type	
	No POS	POS	No POS	POS
Precision	70.83	71.81	74.02	73.21
Complete match	15.87	20.00	23.37	25.45

When there are multiple parse trees for a sentence (which is very often the case even with short sentences), we choose one of *the derivation trees whose derived trees have smallest number of nodes* because these parses correspond to the most specific tree.[13]

5.1 *Performance without POS tagging*

First, the parser is evaluated without using a POS tagger. That is, the module **vnTagger** is not integrated into the parser. In this setting, each word occurrence of an input sentence is tagged with all possible tags that have been assigned to it in the training set. Unknown words are tagged as common nouns (label `N`). We first evaluate the performance of the constituency analysis. The results are shown in Table 5.[14]

In addition to the familiar precision and recall ratios, other measures are reported to help analyze the results:[15]

[13] In case of equality by this criterion, we take the first result returned by the parser.

[14] The presented evaluation results are calculated automatically by EVALB, a tool used frequently for the evaluation of syntactic constituency analysis which is distributed freely at `http://nlp.cs.nyu.edu/evalb/`.

[15] The F-measure is the harmonic mean of precision and recall and is computed as $F = 2\frac{PR}{P+R}$.

- Complete match ratio is the percentage of sentences where recall and precision are both 100%. About 13% of the test sentences match completely. The complete match ratio for sentences of 10 words or less is 17.57%.
- The average crossing ratio is the number of constituents crossing a test constituent divided by the number of sentences of the test corpus.
- The no crossing ratio is the percentage of sentences which have zero crossing brackets. There are 23% of the test sentences that do not have any crossing (29.73% for the sentences of 10 words or less). There are 55% (respectively 68.92%) of the test sentences which have fewer than three crossings.
- The tagging accuracy is the percentage of correct POS tags (without punctuations). It is interesting to note that the tagging accuracy declines slightly when shorter test sentences are used.

The performance of dependency analysis is evaluated in two versions: with and without type. In the first version, two typed dependencies $type_1(u_1, v_1)$ and $type_2(u_2, v_2)$ are considered equal if three corresponding parts of these dependencies are all equal, that is $type_1 \equiv type_2, u_1 \equiv u_2, v_1 \equiv v_2$. In the second version, we compare only two pairs of concerned words without using their dependency types. The D-accuracy of the two evaluations are given in Table 6.[16] Table 7 shows the system's performance for each dependency type.

We see that the parser works perfectly on coordination structures, as they are inherently unambiguous in both the grammar and the extraction algorithm. The performance on the dependencies of type *argument* is much better than that of type *modifier*. These results justify a higher ambiguity of the adjunction operation of the LTAG formalism (which is related to auxiliary trees) in comparison with the substitution operation (which is related to initial trees).

We observe that the parser could not parse about 16.6% of the test corpus. We believe that there may be two main reasons that some sentences can not be analysed. First, there is an insufficient coverage of the underlying LTAG grammar used by the parser. That is, the gram-

[16] Note that when evaluating the accuracy of a dependency analysis, we do not need to compute precision or recall ratios since they are equal: the number of relations given by the parser always matches the number of correct relations.

Table 7: Performance of dependency analysis by type without or with POS tagging

Type	Precision		Recall		*F*-measure	
	No POS	POS	No POS	POS	No POS	POS
arg	87.57	87.18	79.02	80.95	83.08	83.95
coord	100.00	100.00	100.00	100.00	100.00	100.00
modA	48.57	59.09	62.96	65.00	54.84	61.90
modC	46.67	66.67	43.75	60.00	45.16	63.16
modE	50.00	35.71	56.52	35.71	53.06	35.71
modL	72.73	100.00	47.06	50.00	57.14	66.67
modM	80.00	81.82	53.33	75.00	64.00	78.26
modN	50.00	58.54	66.67	68.57	57.14	63.16
modR	64.10	47.06	60.98	42.11	62.50	44.44
modV	52.63	58.33	62.50	87.50	57.14	70.00

mar extracted from the training corpus does not contain the syntactic structure (elementary trees) of a given sentence to be parsed. Secondly, our heuristic choice of tagging all the new words as a common noun may effectively introduce errors prior to the analysis, which may result in analysis failures. We have not yet thoroughly investigated these causes.

The ambiguity and the duration of parsing are strongly dependent on the length of sentences, as shown in Figure 15. It seems that the number of parses has an exponential growth with respect to the length of the sentence.[17]

5.2 *Performance with POS tagging*

The results reported in the previous subsection make possible a preliminary evaluation of the grammar and the performance of the parser. Nevertheless, the condition under which the experimentation is carried out is rather harsh since the parser has to try all possible syntactic categories of each word of an input sentence. The experiments in this subsection are closer to real use conditions, in that each sentence is first processed by a tagger to remove POS-tagging ambiguity – each word is assigned a unique tag. We have thus a sole sequence of

[17] For some considerably long sentences, the parser could not give any result after a fixed time-out predefined at 3 minutes. We limit the sentence length to 15 words in the experiments with the symbolic parser.

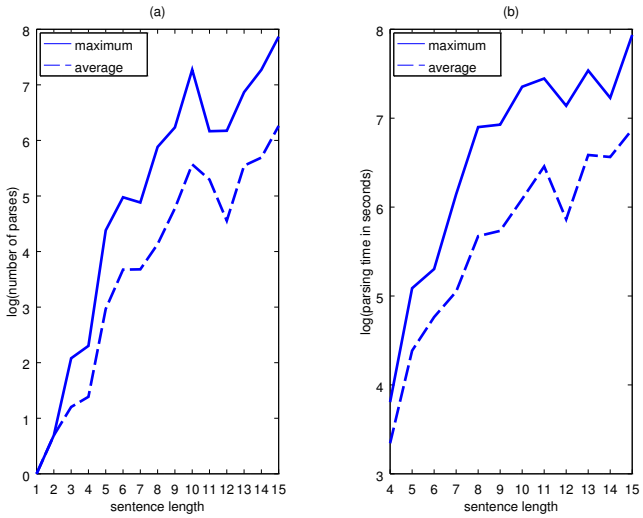


Figure 15: The ambiguity (a) and duration (b) of analysis, average and maximum, according to the length of sentences

words/tags and it is used as input to the syntactic parser. The tagging is done by the **vnTagger** module.

We proceed with the evaluation of this parser version in a similar way as presented for the previous version without POS tagging. We first give constituency parsing results, then dependency parsing results and finally the ambiguity and duration of the parsing.

The T-accuracy of the system is shown in Table 5. By integrating a POS tagger, the tagging accuracy is greatly improved, from 87.72% to 95.25%.[18] This helps improve all the scores of the system, notably the complete match ratio, from 13.00% to 16.67% (and that for sentences of length 10 words or less improves to 20.69%).

The dependency analysis performance both with and without type is shown in Table 6 and the performance of particular dependency types is shown in Table 7.

We see that the performance of the system is improved slightly in comparison with the system without tagging. However, the most important benefit of the parser with the integrated tagger is a strong reduction of analysis ambiguity and time, shown in Figure 16. The tag-

[18] Recall that the test corpus only contains sentences of 30 words or less.

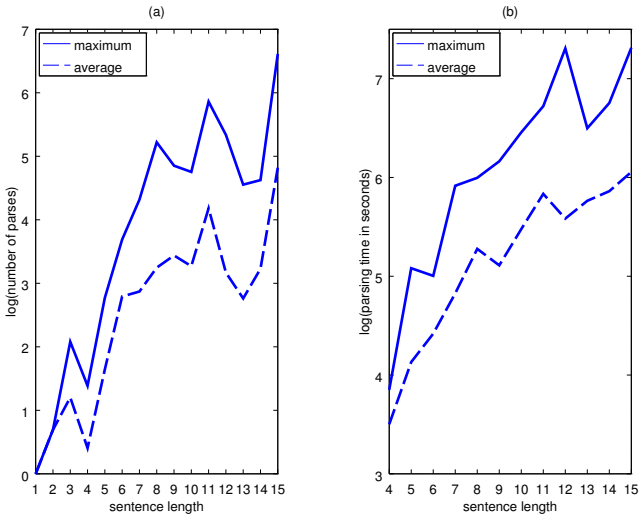


Figure 16: The ambiguity (a) and duration (b) of analysis, average and maximum, with an integrated tagger

ger helps reduce analysis ambiguity fivefold on average and reduces analysis duration three times in comparison with the required time of the parser without prior tagging. Nevertheless, we observe that the integration of the tagger results in a higher number of sentences that the parser could not parse, to 40% of the test corpus. This result is to be expected because in this version the parser uses only a syntactic category (the most probable POS) given by the tagger for each word. (We note also that the precision of the tagger at sentence level is about 32% (Le-Hong *et al.* 2010); that is, the tagger can give correct tags for all the words of a sentence to be parsed only one third of the time).

5.3 *Discussion*

In the previous section, we evaluated a syntactic analysis system based on LTAG for Vietnamese. The best results obtained are 73.21% (dependency accuracy, or D-accuracy) and 69.33% (F-measure of constituency accuracy, or T-accuracy, measured by EVALB) on a test corpus.

It is worth noting that these are the first results of syntactic analysis of Vietnamese based on LTAG. To our knowledge, to date there

have been few published works on the syntactic analysis of Vietnamese. The most complete report on parser performance for Vietnamese is an empirical study of applying probabilistic CFG parsing models by Collins (2003); its best result on constituency analysis is 78% T-accuracy on a test corpus, while there is no result reported for dependency analysis. Concerning the constituency parsing result, their parser is slightly better than ours. However, these results are not directly comparable since the parsing models are trained and tested on a different corpus.

Our first results of the syntactic parsing of Vietnamese are rather good although they are still significantly weaker than parsing results for well-studied languages like English (whose T-accuracy is 91.10% (Carreras *et al.* 2008) and whose D-accuracy is 92.93% (Koo and Collins 2010) on the Penn Treebank) or French (T-accuracy is 86.41% (Candito *et al.* 2009a) and D-accuracy is 85.55% on a French treebank (Candito *et al.* 2010)). However, we can improve our results by correcting three main sources of errors identified by the experiments; we examine each such type of error presently.

The principal source of parsing errors is the selection of parse. When there are multiple parses for a sentence, only the parse whose derivation tree contains the fewest nodes is selected. Although the returned tree corresponds to the most specific analysis, it is obvious that this selection method is purely heuristic and fragile. However, the use of a probabilistic parser does not improve significantly the parsing accuracy. We think that the parameters of the statistical parser are currently not optimised for parsing Vietnamese or the Vietnamese grammar is not large enough in order for the statistical parser to be effective. Consequently, optimising parameters of the statistical parsing model could help improve the parsing performance.

The second source of parsing errors is the POS tagging. In the experiments with a tagger integrated, we use only the most confident prediction generated by **vnTagger** as input to the parser. We have seen that the tagger often makes errors at the sentence level; perfectly tagged sentences are rare. A tagging error may effectively introduce one or more parsing errors. An improvement in tagging performance is thus another necessary condition to improve the performance of the parser.

The third source of parsing errors concerns the coverage of the grammar used in the experiments. In general, the proportion of test sentences having at least one word that the grammar does not recognize is rather high, at about 15%. Consequently, the parser could not build the correct analysis for these sentences. A straightforward solution to this problem is to enlarge the coverage of the LTAG grammar, which in turn necessitates an enlargement of the Vietnamese treebank. However, developing such a corpus is an expensive and labour-intensive task. In addition, this may lead to the typical problem of a symbolic syntactic parser: the tradeoff between its performance and its efficiency. This is an interesting problem in itself, which we shall investigate in future works.

6 CONCLUSION

In this article, we have presented a complete syntactic component for Vietnamese language processing. The component comprises two essential resources: a lexicalized tree-adjoining grammar for Vietnamese and a set of software tools that are chained together to produce syntactic structures from Vietnamese raw text. The grammar is extracted automatically from a treebank by an efficient algorithm. The software includes necessary modules for detecting sentence boundaries, tokenizing word units, part-of-speech tagging and syntactic parsing. This syntactic component is the first system capable of generating both constituency and dependency analyses for the language with encouraging performance.

Syntactic dependency representation of natural sentences has gained a wide interest in the natural language processing community and has been successfully applied to many problems and applications such as machine translation (Ding and Palmer 2004), ontology construction (Snow *et al.* 2005) and automatic question answering (Lin and Pantel 2001). A primary advantage of dependency representation is its natural mechanism for representing discontinuous constructions or long distance dependencies which are common in Vietnamese. We think that the presence of a good dependency schema and a dependency parser for Vietnamese will be very helpful in a wide range of tasks for Vietnamese processing.

We have seen in recent years a rapid increase of research on data-driven dependency parsers, especially the rise of statistical methods in natural language processing where dependency annotated corpora exist. These parsers use one of two predominant paradigms for data-driven dependency parsing which are often called graph-based and transition-based dependency parsing. However, the constituency parser and dependency parser developed in this work are currently purely symbolic in that they do not make use of any probabilistic evidence to discriminate good parses from bad ones for a given sentence, regardless of its grammaticality. An initial investigation of statistical dependency parsing for Vietnamese has shown encouraging results (Nguyen *et al.* 2013). We believe that there is room to improve the performance of dependency parsers in general and of our dependency parser in particular by employing a hybrid approach: use elementary trees of an lexicalized tree-adjoining grammar as good syntactic features in a statistical dependency parser. This is an interesting problem that we plan to work on in the future.

ACKNOWLEDGEMENTS

This research is funded by the Vietnam National University, Hanoi (VNU) under project number QG.15.04. Any opinions, findings and conclusion expressed in this paper are those of the authors and do not necessarily reflect the view of VNU.

We are grateful to our three anonymous reviewers for their insightful comments, which helped us improve the quality of the article in terms of both presentation and content. Finally, we thank the copy editors of the Journal of Language Modelling for their great job on the manuscript.

REFERENCES

Anne ABEILLÉ, Lionel CLÉMENT, and François TOUSSENEL (2003), Building a treebank for French, in Anne ABEILLÉ, editor, *Treebanks: Building and Using Parsed Corpora*, volume 20 of *Text, Speech and Language Technology*, pp. 165–187, Springer Netherlands.

Mark ALVES (1999), What's so Chinese about Vietnamese?, in *Proceedings of the Ninth Annual Meeting of the Southeast Asian Linguistics Society*, pp. 221–224, University of California, Berkeley, USA.

Jens BÄCKER and Karin HARBUSCH (2002), Hidden Markov model-based supertagging in a user-initiative dialogue system, in *Proceedings of TAG+6*, pp. 269–278, Universita di Venezia, Italy.

Marie CANDITO, Benoît CRABBÉ, and Djamé SEDDAH (2009a), On statistical parsing of French with supervised and semi-supervised strategies, in *Proceedings of EACL 2009 Workshop on Computational Linguistic Aspects of Grammatical Inference*, pp. 49–57, Athens, Greece.

Marie CANDITO, Benoît CRABBÉ, and Pascal DENIS (2010), Statistical French dependency parsing: treebank conversion and first results, in *Proceedings of LREC 2010*, pp. 19–21, Valletta, Malta.

Marie CANDITO, Benoît CRABBÉ, Pascal DENIS, and François GUÉRIN (2009b), Analyse syntaxique du français : des constituants aux dépendances (Syntactic Parsing of French: from constituents to dependencies), in *Actes de Traitement Automatique des Langues*, pp. 40–49, Senlis, France.

John CAROLL, Ted BRISCOE, and Antonio SANFILIPPO (1998), Parser evaluation: a survey and a new proposal, in *Proceedings of LREC 1998*, Granada, Spain.

Xavier CARRERAS, Michael COLLINS, and Terry KOO (2008), TAG, dynamic programming, and the perceptron for efficient, feature-rich parsing, in *Proceedings of CoNLL 2008*, pp. 9–16, Manchester, UK.

John CHEN, Srinivas BANGALORE, and K. VIJAY-SHANKER (2006), Automated extraction of tree-adjoining grammars from treebanks, *Natural Language Engineering*, 12(3):251–299.

John CHEN and K. VIJAY-SHANKER (2000), Automated extraction of TAGs from the Penn treebank, in *Proceedings of the Sixth International Workshop on Parsing Technologies*.

David CHIANG (2000), Statistical parsing with an automatically extracted tree adjoining grammar, in *Proceedings of ACL*, pp. 456–463, Morristown, New Jersey, USA.

Michael COLLINS (1997), Three generative, lexicalised models for statistical parsing, in *Proceedings of ACL*, pp. 16–23, Association for Computational Linguistics, Stroudsburg, Pennsylvania, USA.

Michael COLLINS (2003), Head-driven statistical models for natural language parsing, *Computational Linguistics*, 29(4):589–637.

Benoît CRABBÉ, Denys DUCHIER, Claire GARDENT, Josheph Le ROUX, and Yannick PARMENTIER (2013), XMG: eXtensible MetaGrammar, *Computational Linguistics*, 39(3):591–629.

Marie-Catherine DE MARNEFFE, Bill MACCARTNEY, and Christopher D. MANNING (2006), Generating typed dependency parses from phrase structure parses, in *Proceedings of LREC 2006*, pp. 449–454, Genoa, Italy.

Yuan DING and Martha PALMER (2004), Synchronous dependency insertion grammars: a grammar formalism for syntax-based statistical machine translation, in *Workshop on Recent Advances in Dependency Grammars*, pp. 90–97, Geneva, Switzerland.

Robert FRANK (2002), *Phrase structure composition and syntactic dependencies*, MIT Press, Boston, USA.

Nizar HABASH and Owen RAMBOW (2004), Extracting a tree-adjoining grammar from the Penn Arabic treebank, in *Actes de Traitement Automatique des Langues*, pp. 50–55, Fez, Morocco.

Cao Xuân HẠO (2000), *Vietnamese – Some Questions on Phonetics, Syntax and Semantics (in Vietnamese)*, NXB GD, Hanoi, Vietnam.

Phê HOÀNG (2002), *Vietnamese Dictionary*, NXB DN, Danang, Vietnam.

Đạt HỮU, Trí Dõi TRẦN, and Thanh Lan ĐÀO (1998), *Basis of Vietnamese (in Vietnamese)*, NXB GD, Hanoi, Vietnam.

Ane-Dybro JOHANSEN (2004), *Extraction des grammaires LTAG à partir d'un corpus étiquetté syntaxiquement*, Master's thesis, Université Paris 7, Paris, France.

Richard JOHANSSON and Pierre NUGUES (2008), Dependency-based syntactic-semantic analysis with PropBank and NomBank, in *CoNLL 2008: Proceedings of the Twelfth Conference on Computational Natural Language Learning*, pp. 183–187, Manchester, UK.

Aravind K. JOSHI and Yves SCHABES (1997), Tree Adjoining Grammars, in Grzegorz ROZENBERG and Arto SALOMAA, editors, *Handbooks of Formal Languages and Automata*, pp. 69–123, Springer-Verlag, New York, USA.

Miriam KAESHAMMER (2012), *A German treebank and lexicon for tree-adjoining grammars*, Master's thesis, Universitat des Saarlandes, Saarlandes, Germany.

Laura KALLMEYER and Marco KUHLMANN (2012), A formal model for plausible dependencies in lexicalized tree adjoining grammar, in *Proceedings of TAG+11*, pp. 108–116, Paris, France.

Tracy Holloway KING, Richard CROUCH, Stefan RIEZLER, Mary DALRYMPLE, and Ronald M. KAPLAN (2003), The PARC 700 dependency bank, in *Proceedings of 4th International Workshop on Linguistically Interpreted Corpora*, pp. 1–8, Budapest, Hungary.

Terry KOO and Michael COLLINS (2010), Efficient third-order dependency parsers, in *Proceedings of ACL*, pp. 1–11, Uppsala, Sweden.

Sandra KÜBLER, Ryan MCDONALD, and Joakim NIVRE (2009), *Dependency parsing*, Morgan & Claypool Publishers.

Phuong LE-HONG, Thi Minh Huyen NGUYEN, Azim ROUSSANALY, and Tuong Vinh HO (2008), A hybrid approach to word segmentation of Vietnamese texts, in *Proceedings of LATA, LNCS 5196*, pp. 240–249, Springer.

Phuong LE-HONG, Azim ROUSSANALY, Thi Minh Huyen NGUYEN, and Mathias ROSSIGNOL (2010), An empirical study of maximum entropy approach for part-of-speech tagging of Vietnamese texts, in *Actes de Traitement Automatique des Langues*, pp. 50–61, Montreal, Canada.

Charles N. LI and Sandra A. THOMPSON (1976), Subject and topic: a new typology of language, in *Subject and topic*, pp. 457–489, London/New York: Academic Press.

Dekang LIN and Patrick PANTEL (2001), Discovery of inference rules for question answering, *Natural Language Engineering*, 7(4):343–360.

David M. MAGERMAN (1995), Statistical decision-tree models for parsing, in *Proceedings of ACL*, pp. 276–283, Stroudsburg, Pennsylvania, USA.

Ryan MCDONALD, Joakim NIVRE, Yvonne QUIRMBACH-BRUNDAGE, Yoav GOLDBERG, Dipanjan DAS, Kuzman GANCHEV, Keith HALL, Slav PETROV, Hao ZHANG, Oscar TÄCKSTRÖM, Claudia BEDINI, Nuria Bertomeu CASTELLÓ, and Jungmee LEE (2013), Universal dependency annotation for multilingual parsing, in *Proceedings of ACL*, pp. 92–97, Sofia, Bulgaria.

Ryan MCDONALD and Fernando PEREIRA (2006), Online learning of approximate dependency parsing algorithms, in *Proceedings of EACL*, pp. 81–88, Trento, Italy.

Alexis NASR (2004), *Analyse syntaxique probabiliste pour grammaires de dépendances extraites automatiquement*, Habilitation à diriger des recherches, Université Paris 7, Paris, France.

Günter NEUMANN (2003), A uniform method for automatically extracting stochastic lexicalized tree grammar from treebank and HPSG, in Anne ABEILLÉ, editor, *Treebanks: Building and Using Parsed Corpora*, volume 20 of *Text, Speech and Language Technology*, pp. 351–365, Springer Netherlands.

Phuong Thai NGUYEN, Luong Vu XUAN, Thi Minh Huyen NGUYEN, Van Hiep NGUYEN, and Phuong LE-HONG (2009), Building a large syntactically-annotated corpus of Vietnamese, in *Proceedings of the 3rd Linguistic Annotation Workshop, ACL-IJCNLP*, pp. 182–185, Suntec City, Singapore.

Thi Luong NGUYEN, My Linh HA, Viet Hung NGUYEN, Thi Minh Huyen NGUYEN, and Phuong LE-HONG (2013), Building a treebank for Vietnamese dependency parsing, in *The 10th IEEE RIVF*, pp. 147–151, IEEE, Hanoi, Vietnam.

Thi Minh Huyen NGUYEN, Laurent ROMARY, Mathias ROSSIGNOL, and Xuan Luong VU (2006), A lexicon for Vietnamese language processing, *Language Resources and Evaluation*, 40(3–4).

Jaokim NIVRE and Ryan MCDONALD (2008), Integrating graph-Based and transition-Based dependency parsers, in *Proceedings of ACL-08*, pp. 950–958, ACL, Columbus, Ohio, USA.

Joakim NIVRE (2003), An efficient algorithm for projective dependency parsing, in *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT 03)*, pp. 149–160, Nancy, France.

Jungyeul PARK (2006), Extraction of tree adjoining grammars from a treebank for Korean, in *Proceedings of COLING-ACL Student Research Workshop*, pp. 73–78, Morristown, New Jersey, USA.

Patrick PAROUBEK, L. G. POUILLOT, I. ROBBA, and Anne VILNAT (2005), EASY : Campagne d'évaluation des analyseurs syntaxiques (EASY Evaluation compagne of syntactic parsers), in *Actes de Traitement Automatique des Langues*, pp. 3–12, Dourdan, France.

Lewis M. PAUL, Gary F. SIMONS, and Charles D. Fennig (EDS.) (2014), *Ethnologue: Languages of the World, Seventeenth edition*, SIL International, Dallas, Texas, USA.

Owen RAMBOW and Aravind JOSHI (1994), A formal look at dependency grammars and phrase-structure grammars, with special consideration of word-order phenomena, in *Current Issues in Meaning-Text Theory*, pp. 1–20, Pinter, London, UK.

Azim ROUSSANALY, Benoît CRABBÉ, and Jérôme PERRIN (2005), Premier bilan de la participation du LORIA à la campagne d'évaluation EASY, in *Actes de Traitement Automatique des Langues*, pp. 49–52, Dourdan, France.

Yves SCHABES (1990), *Mathematical and computational aspects of lexicalized grammars*, Ph.D. thesis, University of Pennsylvania, Pennsylvania, USA.

Rion SNOW, Dan JURAFSKY, and Andrew Y. NG (2005), Learning syntactic patterns for automatic hypernym discovery, in *Advances in Neural Information Processing Systems*, pp. 1297–1304, Vancouver, Canada.

VIETNAM COMMITTEE ON SOCIAL SCIENCES, editor (1983), *Vietnamese Grammar (in Vietnamese)*, NXB KHXH, Hanoi, Vietnam.

Fei XIA (2001), *Automatic grammar generation from two different perspectives*, Ph.D. thesis, University of Pennsylvania, Pennsylvania, USA.

Fei XIA, Martha PALMER, and Aravind JOSHI (2000), A uniform method of grammar extraction and its applications, in *Proceedings of the joint SIGDAT conference on empirical methods in NLP and very large corpora*, pp. 53–62, Morristown, New Jersey, USA.